Password Factory Documentation

Jordan Bassett & Lizbeth Garcia-Lopez

CS236 - Advanced OOP

Last Updated: March 18, 2025

[1. Project Overview 1](#_Toc193144875)

[2. Features & Functionality 3](#_Toc193144881)

[3. How to Play 3](#_Toc193144882)

[Password Factory Game 3](#_Toc193144883)

[Password Lab 4](#_Toc193144884)

[4. Class Overview 4](#_Toc193144885)

[5. Class Descriptions, Variables, and Methods 4](#_Toc193144886)

[6. Inheritance, Polymorphism, and Interfaces 7](#_Toc193144887)

[7. File Handling 7](#_Toc193144888)

[8. Future Improvements 8](#_Toc193144889)

[9. UML Diagram 9](#_Toc193144890)

[11. Screenshots 10](#_Toc193144891)

[12. Credits 11](#_Toc193144892)

[13. Build & Run Information 12](#_Toc193144893)

[14. Project Repository 12](#_Toc193144894)

# 1. Project Overview

Password Factory is a JavaFX-based educational game developed as a team project by Jordan Bassett and Lizbeth Garcia-Lopez for CS236 - Advanced Object-Oriented Programming at Columbia Basin College. It was written in Java version 23.02 using the NetBeans IDE 24.

The project follows the CS236 Final Project: Strong Password Creation Game for Kids assignment guidelines. The objective was to design and implement a game that teaches users how to create strong, non-dictionary passwords in an interactive and engaging way.

The complete project source code is available on GitHub: <https://github.com/Flareprime/PasswordFactory>

## Project Purpose

The goal of Password Factory is to teach children (and anyone new to password security) how to create strong passwords using memorable phrases, randomness, and variety in character types.

The game provides real-time feedback, encourages experimentation, and penalizes weak passwords (e.g., short or common words).

This project fulfills all requirements across the four phases outlined in the assignment:

## Phase 1 Requirements Met:

* Designed and implemented a Graphical User Interface (GUI) using JavaFX.
* Created multiple classes, including at least one abstract class (PasswordComponent) and one interface (PasswordChecker).
* Demonstrated inheritance and object composition in class design.
* Produced a UML diagram aligning with the final implementation.

## Phase 2 Requirements Met:

* Implemented all class methods for password evaluation and hint generation.
* Included constructors for all subclasses.
* Provided console-based testing using PasswordValidatorTest and PasswordFactoryConsole during early development phases.
* Submitted screenshots of program outputs and tested features.

## Phase 3 Requirements Met:

* Implemented plain text file handling for phrases.txt and common\_words.txt
* Implemented random access binary handling for password\_log.bin
* Provided user-editable lists of phrases and common words within the Password Lab interface.
* Ensured data persistence through file saving/loading, as required.
* Provided testing and screenshots of file processing features.

## Phase 4 Requirements Met:

* Fully functional application with two screens:
  + Password Factory Game
  + Password Lab
* Provided gameplay GUI with timers, hints, scoring, and phrase management.
* Demonstrated the second window (Password Lab) for testing and file editing.
* Provided screenshots of gameplay and password creation results.
* Highlighted and documented class relationships, file handling, password creation logic, and GUI components.

# 2. Features & Functionality

* Password Strength Evaluation
  + Length, uppercase, lowercase, numbers, special characters
  + Detects common words, which are loaded from a file that the user can add to, either within the program or outside.
  + Suggests improvements with dynamic hints
* Difficulty Levels
  + Easy (60 seconds), Medium (45 seconds), Hard (30 seconds)
  + Difficulty impacts time and required password complexity
* Helpful Phrases List
  + User-editable phrases to inspire stronger passwords
  + Phrases can be added in both the **Game** and the **Password Lab**
* Common Words Editing
  + Users can add/remove common words directly from the Password Lab
  + Changes update the common word detection in real time
* Password Logging
  + Stores passwords and their evaluated strength
  + Saves logs to a file for review
* File Handling
  + Loads and saves common words and phrases from plain text files. Plain text allows users to edit the files outside of the program, allowing for bulk imports from outside sources.
* Hint System
  + Real-time dynamic suggestions to strengthen passwords
* Clean JavaFX UI
  + Menu screens
  + Game and lab screens
  + About screen with system information

# 3. How to Play

## Password Factory Game

1. Choose a difficulty level from the **Game Setup** screen.
2. Create strong passwords in the allotted time.
3. Submit passwords for evaluation.
4. Get instant feedback and improve!
5. At the end of the round, review your generated passwords and their scores.
6. Add helpful phrases on the fly, which are saved and available for future rounds.

### Difficulty Levels and Scoring Table

This table summarizes how different difficulty levels affect gameplay and scoring:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Difficulty | Timer | Common Word Penalty | Min Points for Moderate | Other Notes |
| Easy | 60s | -2 points | 3 points | Hints are generous |
| Medium | 45s | -3 points | 4 points | Less time, stricter scoring |
| Hard | 30s | -4 points / auto-fail | 5 points | Requires complexity and fast typing |

* Password strength scoring considers length, character variety, and use of common words.
* Passwords below the Moderate threshold are marked Weak and receive improvement hints.
* Strong passwords exceed the Moderate threshold by 2 points or more.

## Password Lab

1. Access from the Main Menu.
2. Enter and test any password you like.
3. A button to generate a password using a combination of the given phrases and random characters/numbers.
4. View a password’s strength and get hints for improvement.
5. Edit common words and helpful phrases:

* Add/remove helpful phrases (saved to phrases.txt).
* Add/remove common words (saved to common\_words.txt).

1. These updates are immediately used in both the Lab and the Game.

## 4. Class Overview

|  |  |
| --- | --- |
| **Class** | **Purpose** |
| App | Launches the application, controls screen switching. |
| PasswordGame | Runs the main game loop, handles scoring and timing. |
| PasswordLab | Sandbox mode for testing passwords and editing files. |
| GameSetupScreen | Provides difficulty selection and instructions. |
| AboutScreen | Displays app and system information. |
| SystemInfo | Utility for retrieving Java and JavaFX versions. |
| PasswordChecker | Interface for password evaluation methods. |
| PasswordComponent | Abstract class for password evaluation. |
| PasswordValidator | Implements password strength evaluation and feedback. |
| Hint | Represents a feedback message for the user. |
| HintManager | Manages dynamic and generic hints. |
| PasswordFileManager | Handles reading/writing phrases and common words to files. |
| PasswordValidatorTest | Unit tests for the password validator (console-based). |
| PasswordFactoryConsole | Console version of password testing (manual testing). |

## 5. Class Descriptions, Variables, and Methods

**AboutScreen**

Displays the "About" window, providing project details and system information.

* aboutStage - The pop-up window for the About screen
* show() - Opens the About screen window.
* Reason to use SystemInfo.java that Netbeans always includes.
* Doubled as a sandbox window during development to test JavaFX layouts.

**App (Main Application Class)**

Manages screen switching between PasswordGame, PasswordLab, GameSetupScreen, and AboutScreen.

* start(Stage) - Initializes the application and starts the main stage (window).
* showGame() - Displays the PasswordGame screen.
* showLab() - Displays the PasswordLab screen.

**GameSetupScreen**

Explains the game and allows difficulty selection before starting PasswordGame.

* private Stage mainMenuStage – Stores a reference to the main menu window for quick switching between modes.
* public GameSetupScreen(Stage mainMenuStage) – Constructor that accepts the main menu stage
* public void show() – Displays the setup window, lets the player pick a difficulty, and starts the game
* "Start Game" → Opens PasswordGame.java with the selected difficulty.
* "Back to Main Menu" → Closes setup and reopens App.java

**Hint**

Holds individual hint data (message and category) for use in password strength feedback.

* message : String - The actual hint text (e.g., "Use at least 8 characters.").
* category : String - The category of the hint (e.g., "Length", "Complexity").
* getHint() - Returns the hint message.
* getCategory() - Returns the category of the hint.

**HintManager**

Manages both dynamic and static hints, providing them to PasswordGame and PasswordLab as needed.

* dynamicHints : List<Hint> - Stores temporary, dynamically generated hints during gameplay.
* genericHints : List<Hint> - Stores static, preloaded hints for tutorials.
* addDynamicHint(Hint) - Adds a dynamically generated hint during gameplay.
* getGenericHints() - Retrieves the list of generic hints.
* getDynamicHints() - Retrieves the list of dynamic hints for the current round.
* clearDynamicHints() - Clears the dynamic hints after a round ends.

**PasswordChecker (Interface)**

Defines the structure for validating passwords and generating feedback.

* checkStrength() - Validates the strength of a given password.
* giveFeedback() - Provides feedback or hints about the weaknesses of the password.

**PasswordComponent (Abstract Class)**

Superclass for password-related logic. Implements PasswordChecker

* checkStrength(password: String) : int - Abstract method to evaluate password strength.
* giveFeedback(password: String) : String - Abstract method to provide feedback.
* containsCommonWord(password: String, commonWords: String[]) : boolean - **Helper method** to check for common weak words.

**PasswordFactoryConsole <<Console App for Testing and Debugging>>**

A console-based version of Password Factory, developed early in the project to prototype password assembly and validation before the GUI was implemented. It allows users to select phrases, numbers, and special characters to generate passwords, which are then evaluated and saved to a file.

* PHRASES : String[] – Sample phrases hardcoded for user selection.
* SPECIAL\_CHARS : String[] – Special characters available for password construction.
* main(args : String[]) : void – Runs the console app, prompts for user input, generates and evaluates a password, and saves it to a file.

**PasswordFileManager**

Manages all file operations for Password Factory, including loading and saving helpful phrases, common words, and logging password submissions.

The project requirement in Phase 3 states: "Store each remembered phrase on disk using a database or a random access file."

After careful consideration, we made a design decision to store helpful phrases and common words in plain text files (phrases.txt and common\_words.txt). This allows users to easily add or modify multiple phrases in bulk without needing to interact directly with the application. We prioritized usability and accessibility, aligning with Columbia Basin College’s Inclusiveness and Accessibility policies, by ensuring the system supports user-friendly data management.

However, recognizing that password logs contain potentially sensitive information, we chose to implement a RandomAccessFile (binary file) for passwordLogFile. This approach ensures that generated passwords are not stored in plaintext and demonstrates proper file handling and security measures. The binary log prevents casual access to sensitive data while fulfilling the Phase 3 requirement of implementing random access file processing.

We applied critical thinking as outlined in [CBC’s Institutional Learning Outcomes](https://www.columbiabasin.edu/learn/discover-your-path/pathways-at-cbc/our-commitment-to-your-learning.html). We used random access file handling where it made the most practical and secure sense, ensuring we addressed file processing requirements while delivering a better user experience.

* phrasesFile : String – File name for helpful phrases.
* passwordLogFile : String – File name for the password log.
* logWriter : PrintWriter – For writing password logs.
* PasswordFileManager(phrasesFile, logFile) – Constructor.
* loadPhrases() : List<String> – Loads helpful phrases.
* savePhrases(phrases, filename) : void – Saves helpful phrases.
* loadCommonWords(filename) : Set<String> – Loads common words.
* saveCommonWords(words, filename) : void – Saves common words.
* addPhrase(phrase) : boolean – Adds a phrase to the list and saves.
* initPasswordLog() : void – Initializes the password log writer.
* savePassword(password, strength) : void – Saves a password and its rating to the log.
* closeLog() : void – Closes the log writer.

**PasswordGame**

Manages the core gameplay logic, including rounds, scoring, and displaying feedback.

* difficulty : String – The selected difficulty level (Easy, Medium, Hard).
* roundTime : int – The total time for the round.
* timeLeft : int – Countdown timer for each round.
* score : int – Tracks the player’s score.
* passwordsSubmitted : int – Tracks how many passwords the player has submitted.
* passwordChecker : PasswordValidator – Validates password strength and provides feedback.
* fileManager : PasswordFileManager – Handles saving phrases, logs, etc.
* difficulty : String – The selected difficulty level (Easy, Medium, Hard).
* roundTime : int – The total time for the round.
* timeLeft : int – Countdown timer for each round.
* score : int – Tracks the player’s score.
* passwordsSubmitted : int – Tracks how many passwords the player has submitted.
* passwordChecker : PasswordValidator – Validates password strength and provides feedback.
* fileManager : PasswordFileManager – Handles saving phrases, logs, etc.

**PasswordLab**

Sandbox screen for testing passwords and editing common words and helpful phrases. Features a generate password button that uses phrases and randomized characters to create a strong password.

* fileManager : PasswordFileManager – Loads and saves common words and phrases.
* validator : PasswordValidator – Validates password strength in lab mode.
* commonWordsList : ListView<String> – Displays editable common words.
* phrasesList : ListView<String> – Displays editable helpful phrases.
* showLab() – Displays the Password Lab window.
* testPassword() – Evaluates a password and provides feedback.
* saveCommonWords() – Saves edits to common\_words.txt.
* savePhrases() – Saves edits to phrases.txt.

**PasswordValidator (Extends PasswordComponent, Implements PasswordChecker)**

Handles password validation and generates detailed feedback.

* difficulty : String – Difficulty level (affects thresholds and penalties).
* commonWordPenalty : int – Penalty applied when a common word is detected.
* moderateThreshold : int – Points required for moderate strength.
* strongThreshold : int – Points required for strong strength.
* getHintManager() – Returns the current HintManager (optional, if exposed).

**PasswordValidatorTest <<Test Class>>**

A console-based test class used to validate the functionality of PasswordValidator during development. Runs a series of sample passwords through the validator to display their strength ratings and feedback.

* All variables are local to main
* main(args : String[]) : void – Tests several sample passwords against the PasswordValidator and prints the results.

**SystemInfo**

Retrieves system details such as the Java and JavaFX version. NetBeans automatically includes this, and it is invoked in the About screen.

* javaVersion() - Returns the Java version as a string.
* javafxVersion() - Returns the JavaFX version as a string.

## 6. Inheritance, Polymorphism, and Interfaces

* PasswordChecker is an interface that defines the contract for gauging password strength and providing feedback. By using this interface, any class that implements it, such as PasswordValidator, standardizes game logic to validate passwords.
* PasswordValidator implements the PasswordChecker interface, providing a specific implementation for checking password strength based on rules like length, character variety, and the detection of common words. This allows the game to assess password quality based on difficulty levels (Easy, Medium, Hard), and adjust validation accordingly.
* PasswordValidator extends the abstract class PasswordComponent, which implements the PasswordChecker interface. This inheritance structure allows PasswordValidator to inherit shared logic from PasswordComponent, such as checking for common weak words using the containsCommonWord helper method, while still providing its own specific logic for password evaluation.
* Polymorphism is a core feature in the design, allowing the game to be flexible and extensible. By using the PasswordChecker interface, the game can work with any class that implements this interface. This means the game can easily switch between different validation strategies without needing to modify the core game logic. For example, PasswordValidator can be used to validate passwords during gameplay, but if a different validation strategy is needed in the future, a new class can implement the PasswordChecker interface without changing how the game operates.
* The use of polymorphism allows the game to dynamically use different classes that implement the PasswordChecker interface. Whether it's for evaluating a simple password in the "Easy" mode or a complex one in "Hard" mode, the game doesn’t need to know the exact implementation of the password validator; it only needs to rely on the methods defined in the interface. This keeps the game flexible, allowing for new validation strategies to be introduced in the future with minimal changes to the rest of the code.
* PasswordComponent serves as an abstract class that enforces common functionality for all password-related logic while leaving specific validation methods abstract. This allows subclasses like PasswordValidator to focus on the specific logic for password strength, while still adhering to the structure laid out by the interface and abstract class. This combination of inheritance and polymorphism ensures that the project is both extensible and maintainable, reducing redundancy and allowing for easy updates or changes to password validation strategies in the future.

## 7. File Handling

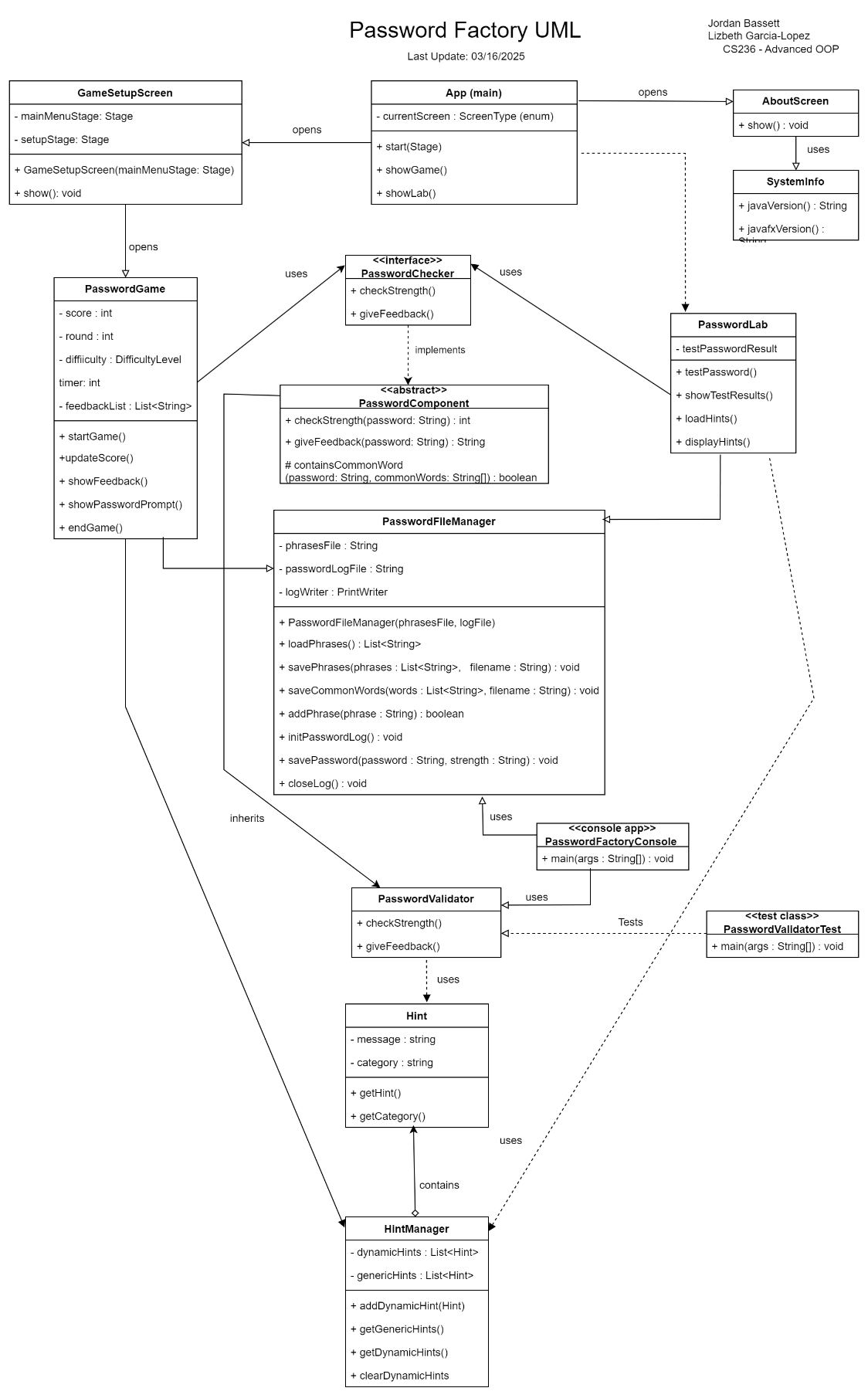
* Loads helpful phrases from phrases.txt.
* Loads common words from common\_words.txt.
* Saves password logs to random access binary file: password\_log.bin.
* Phrases and common words files are saved as plain text for easy editing outside of the program.
* PasswordFileManager manages all file I/O operations.

## 8. Future Improvements

* Add more advanced password strength checks (e.g., pattern detection, dictionary attacks).
* Provide audio/visual feedback (e.g., alarms, animations).
* Implement high score tracking and user accounts.
* Export generated passwords for personal use.
* Refine the hint system with more tailored suggestions.

## 9. UML Diagram

Describes class relationships, including inheritance, interfaces, file handling, and test classes. Updated as the program evolved over time. This diagram was created using [draw.io](https://app.diagrams.net/).



## 11. Screenshots

|  |  |
| --- | --- |
| Start/Main Menu Screen | About Screen |
| Start/Main Menu Screen | About Screen |
|  |  |
| Game Setup | Game Screen |
| Game Setup | Game Screen |
|  |  |
| Game Screen Showing Hints | Game Screen with Password Inputs |
| Game Screen Showing Hints | Game Screen with Password Inputs |
|  |  |
| Game Screen with a Phrase Added | Password Lab Screen |
| Game Screen with a Phrase Added | Password Lab Screen |
|  |  |
| Password Lab Screen with a Generated Password | Password Lab Screen with a Common Word Added and Saved |
| Password Lab Screen with a Generated Password | Password Lab Screen with a Common Word Added and Saved |
|  |  |

12. Credits

Developers:

* Jordan Bassett
* Lizbeth Garcia-Lopez

Course: CS236 - Advanced OOP at Columbia Basin College, Winter Quarter 2025

Instructor: Joshua Bee

## 13. Build & Run Information

Development Environment

* Java Version: OpenJDK 17+
* JavaFX SDK: Version 20
* Build System: Maven 3.8.0
* IDE: Apache NetBeans IDE 24

The files *phrases.tx*t and *common\_words.txt* are stored in the project root directory.

## 14. Project Repository

The complete project source code is available on GitHub:

<https://github.com/Flareprime/PasswordFactory>

The repository includes:

* Source code for Password Factory
* Documentation files
* UML diagram
* README.md and TODO.md